PATENT
Atty. Docket No. 3655-0147P

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	William J. WALKER	Conf. No.:	8223
Appln. No.:	09/837, 929	Group:	2122
Filed:	April 19, 2001	Examiner:	J. RUTTEN
For:	A MECHANISM FOR CONVERTING BETWEEN JAVA CLASSES AND XML		

DECLARATION SUBMITTED UNDER 37 C.F.R. § 1.131

I, William J. WALKER, inventor of the above-captioned application, do hereby declare the following. The present application was fully conceived by me prior to January 26, 2000. Attached are documents fully describing the conception and due diligence leading to constructive reduction to practice of the present invention on April 19, 2001. Specifically:

1. I have reviewed the attached seven (7) page document entitled "A Mechanism For Converting Between Java Classes and XML" (THE DOCUMENT) and confirm that THE DOCUMENT was authored by me.
2. I have reviewed the attached e-mail dated January 26, 2000 and confirm that the e-mail was written by me and forwarded to Mr. Joseph Opalach, Corporate Counsel, Lucent Technologies, attaching THE DOCUMENT.
3. I have reviewed the attached e-mail dated January 27, 2000 from Mr. Opalach and acknowledge that it is an accurate copy of the e-mail I received on January 27, 2000.
4. I have reviewed the "Patent Submission IDS #121717" opened by Mr. Opalach on January 27, 2000 and verify that it relates to my invention described in THE DOCUMENT.

5. I have reviewed the Lucent Technologies Docketing records dated January 31, 2000 and August 3, 2000 referencing my invention submission #121717 and verify that the record correspond to my invention described in THE DOCUMENT.
6. I have reviewed the e-mail dated May 8, 2000 to Mr. Opalach inquiring about status of invention submission #121717 and verified that it was sent by me.
7. I have reviewed the e-mail dated May 8, 2000 from Mr. Opalach and verify that I received the e-mail indicating that consideration of invention submission #121717 would be delayed, but that additional information from me to help prioritize the present invention could accelerate consideration of invention submission #121717 by Lucent Technologies.
8. I have reviewed the e-mail dated September 26, 2000 to Mr. Opalach and verify that I sent this e-mail inquiring about the status of invention submission #121717.
9. I have reviewed the e-mail dated September 26, 2000 from Mr. Opalach and verify that I received this e-mail indicating that the invention submission #121717 had been transferred from Lucent Technologies to Thomas Bean, Patent Counsel, AVAYA, Inc.
10. I have reviewed the e-mail dated October 20, 2000 to Thomas Bean, AVAYA, Inc. and verify that I sent it.
11. I have reviewed the e-mail dated October 20, 2000 from Thomas Bean and verify that I received this e-mail indicating that the invention submission #121717 was currently open, and stating that it was AVAYA's policy to seek appropriate protection for intellectual property.
12. I have reviewed the e-mail dated October 20, 2000 from David S. Mohler, Director of Intellectual Property, AVAYA, Inc. suggesting "a very high priority for this application".

13. I have reviewed the FAX dated February 8, 2001 from Linda K. Krichman, AVAYA, Inc. forwarding THE DOCUMENT to outside counsel Thomason, Moser, & Patterson, LLP, instructing that an application be prepared ASAP for filing in the USPTO.
14. I attest that to the best of my knowledge that a draft application was prepared by AVAYA, Inc.'s outside counsel Thomason, Moser, & Patterson, LLP for my review and approval in March or April 2001.
15. I attest that upon approval by me and AVAYA, Inc.'s Patent Counsel, the application was filed in the USPTO on April 19, 2001 by AVAYA, Inc.'s outside counsel Thomason, Moser, & Patterson, LLP.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further declare that these statement and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

William J Walker 3/3/2005

Signed: William J. WALKER Date

BEST AVAILABLE COPY

Facsimile Cover Sheet

To:	Laura E. Crater
Company:	Thomason, Moser & Patterson
From:	Linda K. Krichman
Company:	Avaya Inc.
Phone:	(732) 817-4086
Fax:	(732) 817-4504
Date:	February 8, 2001
Pages including this cover page:	2

RECEIVED
AVAYA IP-LAW - DOCKETING
FEB 7 2001
DOCKET
ROUTE TO

[illegible]

RECOMMENDATION TO FILE
AND REQUEST FOR FILE NO.

FAX COVER SHEET

Date: 2/7/01

To: Outside Counsel Coordinator
Avaya Inc.

Phone: (732) 817-4086
Fax: (732) 817-4504

No. of pages including cover sheet: 1

OC Firm Name/Contact: Eamon J. Wall by Laura E. Crater

Fax No.: 732-530-9808 Phone No.: 732-530-9808

Date Needed: ASAP

Submission File No.: 121717

Recommend Filing: Avaya to provide Application File No. ✓

Recommend No Filing

LIST NAMES OF ALL INVENTORS:

1.	William J. Walker	9.	
2.		10.	
3.		11.	
4.		12.	
5.		13.	
6.		14.	
7.		15.	
8.		16.	

App. Type: ☐ Prov. [✓ New] ☐ CIP ☐ Continuation ☐ Divisional ☐ RCE

First Filed Country (if other than U.S.):

Estimated Fee to File*: $6,900

2/7/01
OK [illegible]

Special Instructions:

Do Not Fill In Below

Avaya's Application File No.: 500007-A-01-US
First-Named Inventor: (WALKER)

*Required if estimated fee is greater than $5500.00

October 2000

```
*********************
***   TX REPORT   ***
*********************

TRANSMISSION OK

TX/RX NO                  3114
CONNECTION TEL                        8174504
SUBADDRESS
CONNECTION ID
ST. TIME                  02/07 15:01
USAGE T                   00'33
PGS. SENT                   1
RESULT                    OK
```

[illegible]

RECOMMENDATION TO FILE AND REQUEST FOR FILE NO.

F A X C O V E R S H E E T

Date: 2/7/01

To: Outside Counsel Coordinator
Avaya Inc.

Phone: (732) 817-4086
Fax: (732) 817-4504

No. of pages including cover sheet: 1

OC Firm Name/Contact: Eamon J. Wall by Laura E. Crater

Fax No.: 732-530-9808 **Phone No.:** 732-530-9808

Date Needed: ASAP

Submission File No.: 121717

Recommend Filing: Avaya to provide Application File No. ✓

Recommend No Filing –

LIST NAMES OF ALL INVENTORS:

1.	William J. Walker	9.	
2.		10.	
3.		11.	
4.		12.	
5.		13.	
6.		14.	
7.		15.	
8.		16.	

[✓ New] ☐ CIP ☐ Continuation ☐ Divisional ☐ RCE

RECOMMENDATION TO FILE AND REQUEST FOR FILE NO.

FAX COVER SHEET

Date: 2/7/01

To:	Outside Counsel Coordinator Avaya Inc.	Phone: Fax:	(732) 817-4086 (732) 817-4504

No. of pages including cover sheet: 1

OC Firm Name/Contact: Eamon J. Wall by Laura E. Crater

Fax No.: 732-530-9808 **Phone No.:** 732-530-9808

Date Needed: ASAP

Submission File No.: 121717

Recommend Filing: Avaya to provide Application File No. ✓

Recommend No Filing _

LIST NAMES OF ALL INVENTORS:

1.	William J. Walker	9.	
2.		10.	
3.		11.	
4.		12.	
5.		13.	
6.		14.	
7.		15.	
8.		16.	

App. Type: ☐ Prov. [✓ New] ☐ CIP ☐ Continuation ☐ Divisional ☐ RCE

First Filed Country (if other than U.S.): ____________

Estimated Fee to File* $6,900

Special Instructions:

Do Not Fill In Below

Avaya's Application File No.,
First-Named Inventor: William J. Walker

*Required if estimated fee is greater than $5500.00

October 2000

A Mechanism for Converting Between Java Classes and XML

Overview

Java and XML (eXtensible Markup Language) technologies provide developers with the tools to write portable code and operate on portable data. Support for XML in the Java platform is increasing with the availability of standard APIs and parsers such as SAX and DOM (Document Object Model). While these APIs provide standard mechanisms for reading XML documents, they work at a relatively low level. Using DOM for example, developers must have a detailed understanding of how to use the API to navigate nodes, elements, attributes, and to extract textual content and then convert the text to useful program data types. This may be tedious, error prone, and requires the developer to work with classes outside the application domain.

This paper describes a mechanism that allows developers to convert easily between Java and XML representations of data while working exclusively with classes from the application domain. With very little work, developers can add XML support for complex hierarchies of any user-defined class, Java primitives (int, float, boolean, etc.) and wrapper classes (Integer, Boolean, Float, etc.) as well as collections and arrays of such objects.

A proposal by Sun Microsystems has been drafted that deals with this topic using a different approach *(An XML Data-Binding Facility for the Java Platform, Mark Reinhold, Core Java Platform Group, 30 July 1999 – see www.javasoft.com/xml)*. They propose a schema compiler that generates Java classes from an XML schema. The API proposed in this paper has the following advantages:

- No schema compiler is required to generate new Java classes.
- New or existing classes can be easily annotated to work with the API.
- The developer has full control and flexibility over how the classes get mapped to XML. Two totally different class implementations can work with the same XML representation in different ways.

The following sections discuss how the API is used including the API class descriptions.

An Example

Data is described in XML in a hierarchical way by tagging elements. An XML document contains one single element (the document element) that may contain any number of other elements. For example, an XML representation of a book store may be written as:

```
<bookStore>
   <name>The Programmer's Book Store</name>
   <address>
      <street>1 Industrial Way</street>
```

```
        <city>Middletown</city>
        <state>NJ</state>
        <zip>07701</zip>
    </address>
    <books>
        <book reviewed="no">
            <title>Xml and Java</title>
            <author>Hiroshi Maruyama</author>
            <cost>$49.00</cost>
        </book>
        <book reviewed="yes">
            <title>Java in a Nutshell</title>
            <author>Flannigan</author>
            <cost>$39.00</cost>
            <review>
                <reviewedBy>Joe</reviewedBy>
                <rating>3.5</rating>
            </review>
            <review>
                <reviewedBy>Bob</reviewedBy>
                <rating>9.5</rating>
            </review>
        </book>
    </books>
</bookStore>
```

The document contains a single element *bookStore*, which has sub-elements for *name*, *address*, and *books*. The *books* element contains a collection of *book* elements. Elements may contain one or more "attributes" such as the *reviewed* attribute of book, used to indicate whether the book has been reviewed or not. Books that have been reviewed contain one or more *review* elements.

This data can be modeled by a set of Java classes, such as:

```
class BookStore {
    String name;
    Address address;
    Vector books;
    // ... public methods
}

class Address {
    String street;
    String city;
    String state;
    String zip;
    // ... public methods
}

class Book {
    String author;
    String title;
    float cost;
    // ... public methods
}

class ReviewedBook extends Book {
    Vector reviews;
    void addReview(Review review);
    Vector getReviews() { return reviews; }
}
```

The goal of the API is to let the developer construct the data using Java only, then somehow convert the classes to XML and later re-construct the contents of the Java classes from the XML file. For example:

```
BookStore bookStore = new BookStore("The Programmer's Book Store");
BookStore.setAddress( new Address("1 Maple St.", "Middletown", "NJ") );

Book book = new Book("Java in a Nutshell", "Flannigan");
Book.setCost(49.0f);
BookStore.add(book);

SaveToXml(bookStore, "bookStore.xml");  // a hypothetical method

// later, read the contents back to Java

BookStore bookStore = ReadFromXml("bookStore.xml");  // a hypothetical method
Collection books = bookStore.getBooks();
```

The next section describes an API that accomplishes this.

XML to Java API

The XmlReaderWriterInterface

In order to convert user-defined types to XML, such as *BookStore*, *Address*, *Book*, and *Review* in the above example, each must include some instructions about how to do the conversion. The API needs to know:

- Which fields inside the Java object should be saved to XML? We may want the Java class to contain other fields that are used internally and should not get converted.
- For each field that we want converted, what tag name should be used when generating the corresponding XML element.
- When reading an XML file and constructing the java objects, what classes should be instantiated for each element? We want to be able to support different classes and different Java implementations using the same XML representation.

The Java to XML API accomplishes this by defining the *XmlReaderWriter* interface. Any class that we wish to convert to XML or construct from an XML document must implement this interface. The interface is defined as follows:

```
interface XmlReaderWriter {
   FieldDescription[] getFieldDescriptions();
   void setAttributes(Hashtable ht);
   Hashtable getAttributes();
}
```

The first method *getFieldDescriptions* is required. This allows the class to define how it should be converted. Typically, this will add just one line of code for all sub-elements contained by the class. The second two methods are optional, and are not used if the class does not use *attributes* as mentioned above. The *FieldDescription* class is described in the next section, but to illustrate it's use, the *BookStore* class from the last example can add the following lines of code to implement it:

```
class BookStore {
   ...
   public FieldDescription[] getFieldDescriptions() {
      return new FieldDescription[] {
         new FieldDescription("name", String.class, "getName", "setName"),
         new FieldDescription("address", Address.class, "getAddress", "setAddress"),
         new FieldDescription("books", Vector.class, "getBooks", "setBooks",
            Book.class, "book"),
      };
   }
   public void setAttributes(Hashtable ht) {;} // not used
   public void Hashtable getAttributes() { return null; } // not used
}
```

Note that the *BookStore* class only needs to describe fields directly contained in it. Sub-elements such as *Address* and *Book* (contained in the collection) will provide their own descriptions by implementing the interface.

The FieldDescription Class

The FieldDescription class provides the set of information needed by the API to convert between Java and XML representations. The FieldDescription class has the following constructors:

```
FieldDescription(String tagName, Class objectClass, String getMethod, String setMethod);

FieldDescription(String tagName, Class objectClass, String getMethod, String setMethod,
   Object contentClasses);

FieldDescription(String tagName, Class objectClass, String getMethod, String setMethod,
   Object contentClasses, Object ContantTagNames);
```

For simple elements, the first constructor is used. When a field is represented as a Collection, Hashtable or an array, the second forms are used to describe the elements in the collection.

For the first form, the parameters are:

- *TagName* – when writing this field to XML, what name should be used for the corresponding XML element tag.
- *ObjectClass* – Specifies the Class to instantiate when constructing this field from XML
- *GetMethod* – The name of the Java method to invoke to retrieve this field.
- *SetMethod* – The name of the Java method to invoke to retrieve this method.

The *contentClass* parameter is used to specify what class of object must be instantiated and constructed from the element. It may be any one of the following:

- A single Class object. In this case, all elements will be represented by the same class.
- The class may be based on element tag name. In this case, the parameter passed in is a Hashtable, where the keys are the element names and the values are the corresponding Class types.
- The class may be based on an attribute value contained in the elements. In this case, the parameter is a Hashtable where they keys are specified in the form *"attrName=attrValue"* and the values are the Class types to use.

A containing class must also specify the element names to use for each field when writing them to the XML document. The *contentName* parameter may be any of the following:

- The same name for all elements. In this case, pass a single String containing the name to use.
- The name may be obtained by invoking a "get" method on the object. In this case, you must specify a method name preceded by an "@" (e.g. "@getMyName"). This method must take no parameters and return a String.
- The name may be based on the class of object. In this case, the keys should be the Class types and the corresponding values should be the tag name to use.
- Based on Hashtable keys, if collection is an instance of java.util.Hashtable. In this case, use the FieldDescription constructor that does not take a contentName parameter.

In order to support inheritance, subclasses only need to define FieldDescriptions for new elements, and simply concatenate the FieldDescriptions of the parent class. For this purpose, the FieldDescription class has a *concat* method to make this easy. For example, the ReviewedBook class inherits from the Book class, so it's *getFieldDescription* method could be written as:

```
class ReviewedBook extends Book {
   ...
   public FieldDescription[] getFieldDescriptions() {
      FieldDescription[] fda = new FieldDescription[] {
         new FieldDescription("reviews", Vector.class, "getReviews", "setReviews",
                  Review.class, "reviews")
      };
      return FieldDescription.concat( fda, super.getFieldDescriptions() );
   }
}
```

For the case of the book collection, we may just construct a Book object for each element if all we are interested in is the base class. However, if we want to construct a different type, depending on the attribute, we may do that as well. We can modify the *contentClass* parameter to be a Hashtable, and specify the type based on attribute:

```
Hashtable ht = new Hashtable();
ht.put("reviewed=yes", ReviewedBook.class);
ht.put("reviewed=no", Book.class);

fd = new FieldDescription("books", Vector.class, "getBooks", "setBooks",
            ht, "book");
```

Now the API can determine what type of class to instantiate based on attribute.

Attributes versus Elements

It is up to an implementation to decide whether to use elements or attributes. For example, consider a User object:

```
User {
   String id;
   String lastName;
   String firstName;
   String phoneNumber
```

}

In this object model, *id*, *lastName*, *firstName* and *phoneNumber* are considered "attributes" of User. In XML, this may be modeled as either:

```
<user>
   <id>1001</id>
   <lastName>Smith</lastName>
   <firstName>Joe</firstName>
   <phoneNumber>732-222-1234</phoneNumber>
</user>
```

or as

```
<user id="12345">
   <lastName>Smith</lastName>
   <firstName>Joe</firstName>
   <phoneNumber>732-222-1234</phoneNumber>
</user>
```

For the second case, rather than describing *id* with a FieldDescription, it would be treated as an attribute in the User class:

```
Class user {
   String id = null;
   ...
   FieldDescription[] getFieldDescriptions() {
      return new FieldDescription[] {
         new FieldDescription("lastName", String.class, "getLastName", "setLastName"),
         new FieldDescription("firstName", String.class, "getFirstName", setFirstName"),
         new FieldDescription("phoneNumber", String.class, "getNumber", "setNumber"),
      };
   }
   Hashtable getAttributes() {
      Hashtable ht = new Hashtable();
      ht.put("id", id);
      return ht;
   }
   void setAttriutes(Hashtable ht) {
      String s = ht.get("id");
      if( s != null ) this.id = new String(s);
   }
}
```

The XmlUtil class

This class provides static methods that load and save Documents to and from XML streams as well as converting Document objects to and from specified Java classes. For the above book store example, the complete code needed to save the BookStore to an XML file and later restore it is as follows:

```
// construct a book store and populate it with books…
BookStore bookStore = new BookStore(…);
Book book = new Book(…);
BookStore.add(book);  // etc.

// turn it into a Document object and save to an XML file
```

```
Document doc = XmlUtil.getDocument("bookStore", bookStore);
XmlUtil.writeXml(doc, "books.xml");

// later, reload the book store from XML
Document doc = XmlUtil.readXml("books.xml");
BookStore bookStore = (BookStore)XmlUtil.getObject(doc, BookStore.class);
Collection books = bookStore.getBooks();  // do something with books
```

The *readXml* and *writeXml* are used to read and write *Documents* to and from XML files (or more generally streams). The conversion from a Java object to XML is accomplished by:

```
Document XmlUtil.getDocument(String docName, Object obj);
```

As long as the top level object and contained objects implement *XmlReaderWriter*, the whole collection can be handled by this call. To convert a Document to any class implementing XmlReaderWriter, use:

```
Object XmlUtil.getObject(Document doc, Class objectClass);
```

An instance of *objectClass* will be instantiated and queried for it's *FieldDescriptions*. From that point the API can determine how to convert all nodes that it encounters. This works recursively through the whole document tree. As mentioned, different object classes can be used to produce different results.

Summary

The API described here is a simple yet powerful mechanism for converting between Java and XML representations. It can be easily applied to any application that is written in Java and is using XML as an external data exchange format. The mechanism will work with any XML parser that implements the standard W3C Document Object model. XML representations are easily converted directly into the Java objects and data types used by developers within their application.

Bean, Thomas J (Tom)

From: Mohler, David S (David)
Sent: Friday, October 20, 2000 12:47 PM
To: Walker, William J (William); Bean, Thomas J (Tom)
Cc: Rudnick, Robert E (Rob); Potkay, Eugene (Gene); Beightol, Dean D (Dean)
Subject: RE: patent candidate

Tom,

I would like to suggest a very high priority on this application. The reason for this is the importance in the industry with anything related to XML or its varients (UXL, VXML, etc). Further, XML is key to several of Avaya's "growth engine" technologies. Bill Gates recently said that "We are betting a significant portion of Microsoft's future on XML." This makes sense because in its ultimate evolution (so far anyway) it helps MS break the strangle-hold that Sun and Linux has on some portions of the operating systems market that MS covets.

During the prosecution of this, you may want to talk with David Volejnicek about potential overlap with Terry Jennings "Interactor" application (Jennings 5) and his new patent idea that he just talked with David about 6 weeks or so (Jennings ?). Getting the broadest possible claims filed in this area would be enormously commercially valuable. I am well versed in this area and would be glad to help in anyway that I can. Please note that this is a very fast moving area and that much of the prior art is only referenced on the web. I would suggest the follwong URLs for prior art research:

http://www.xml.com/xml/pub
http://www.hr-xml.org/
http://www.oasis-open.org/cover/xml.html
http://www.xml-zone.com/
http://www.xmlephant.com/
http://www.extensibility.com/
http://www.xmltree.com/
http://msdn.microsoft.com/xml/default.asp
http://metalab.unc.edu/xml/
http://www.w3.org/XML/Activity.html
http://www.xmlglobal.com/
http://www.perlxml.com/faq/perl-xml-faq.html
http://www.ozemail.com.au/~sakthi/Common/xml.html
http://info.dr.lucent.com/~tdj/docs/xmlaud.doc

Regards -
David

David S. Mohler
Director of Intellectual Property
Avaya Inc.

(303) 538-1093 Voice
(303) 538 -6066 Fax
dmohler@avaya.com email

From: Bean, Thomas J (Tom)
Sent: Friday, October 20, 2000 7:07 AM
To: Walker, William J (William)
Cc: Rudnick, Robert E (Rob); Mohler, David S (David)

Subject: RE: patent candidate

Bill:

This submission is currently open and has not yet been assigned for development of an associated patent application. If you have any further information or details that have emerged since you made your original submission, please forward these to me. Like Lucent, Avaya's policy remains to seek appropriate protection for its intellectual property. We do promote use of our technology within industry standards, for example, by making the technology available under convenient and reasonable licensing terms. I would suggest that we discuss the particulars of your submission with my colleague Rob Rudnick, who has been supporting a number of our Avaya standards teams, to determine how we might proceed in Avaya's best interest.

Tom

Thomas J. Bean
Corporate Counsel
Avaya Inc.
☎ (732) 817-6164 (phone)
☎ (732) 817-4504 (fax)
tjbean@avaya.com

This message is intended only for the designated recipient(s). If you are not a designated recipient, you may not review, copy or distribute this message. If you receive this in error, please notify the sender by reply e-mail and delete this message. Thank you.

From: Walker, William J (William)
Sent: Friday, October 20, 2000 7:37 AM
To: Bean, Thomas J (Tom)
Subject: FW: patent candidate

Hello Tom:

Joseph Opalach said you would be the person to contact about this. Last January, I submitted a candidate for a patent, which deals with a mechanism for converting Java to XML. The submission was accepted, but nothing further was done with it to my knowledge.

Java and XML are very hot technologies right now, and this is a very useful mechanism for converting between the two. If nothing is going to be done regarding a patent, I would really like to see this submitted to Sun's Java Community for consideration as was done for their schema compiler:

http://java.sun.com/aboutJava/communityprocess/jsr/jsr_031_xmld.html

I understand that there are difficulties deciding what patents to go after first in terms of things that can be proven easily. If this one is not easy to go after, I would rather see it dropped as a patent candidate and get it out to the Java community ASAP, in

2

one form or another.

Thanks,
Bill

-----Original Message-----
From: Opalach, Joseph J (Joe)
Sent: Tuesday, September 26, 2000 12:13 PM
To: Walker, William J (William)
Subject: RE: patent candidate

Bill,

Your work was transferred from Lucent to Avaya.

The Avaya patent attorney is Tom Bean.

You'll have to contact Tom for information. However, you might want to wait another couple of weeks, Tom is also responsible for the transfer of (intellectual property) information to Avaya (including, e.g., set up of computers systems, data bases, physical storage issues, etc.) and these next couple of weeks will be hectic for him.

Joe

From: Walker, William J (William)
Sent: Tuesday, September 26, 2000 11:53 AM
To: Opalach, Joseph J (Joe)
Subject: RE: patent candidate

Joe - It's been 4 months, so I just wanted to check back on this ... do you have any further information?

Thanks,
Bill Walker

-----Original Message-----
From: Opalach, Joseph J (Joe)
Sent: Monday, May 08, 2000 3:17 PM
To: Walker, William J (William)
Subject: RE: patent candidate

Bill,

At this point, you will have to be more patient. I do not see anything being done for the next 4 months or so.

With respect to timeframe and someone else coming up with it - that is true for every submission we have.

It would better help me to prioritize this and accelerate it if you could answer the following:

Will the work be a part of a standards contribution ? (At first pass, I don't believe it is but I need to be sure.)

3

How can you detect someone is using your idea - without reverse engineering the website (which implies you would have access to the code) ?

Just fyi, in a very general sense, we prioritize submissions based on how easy it is to get a return on investment for the resulting patent (i.e., a licensing royalty). Patents that pertain to standards or are easy to detect (without incurring the cost of reverse engineering), get to the top of the list faster.

Thanks

Joe

From: Walker, William J (William)
Sent: Monday, May 08, 2000 2:45 PM
To: Opalach, Joseph J (Joe)
Cc: Little, Cheryl (Cheryl)
Subject: RE: patent candidate

I just wanted to check on the status of this patent submission. I have
heard nothing from any attorney as of yet, and it has been over
three months. As you know, both Java and XML are two very hot
technologies today, with new designs coming up daily. My concern
is that if takes much longer to move on this one, that someone else
will come up with this...

Thanks,
Bill Walker

Bill Walker
Lucent Technologies
732-817-4609
wjw@lucent.com

From: Opalach, Joseph J (Joe)
Sent: Thursday, January 27, 2000 11:12 AM
To: Walker, William J (William)
Subject: RE: patent candidate

Hi Bill,

I've looked over your submission and we will move forward with it.

Most of the patent application development is contracted to outside attorneys, so I will get your submission rolling (so-to-speak) to get an outside attorney assigned. I will still be your contact within Lucent if you have any questions, complaints, etc.

4

The outside attorney should be in touch with you in approx. 2 to 3 months. I am going to "schedule" this submission for filing in 6 months (the filing date in the end has to do with the current workload of the assigned attorney and yourself, since you will have to provide assistance and review the patent application, so it could be filed sooner than 6 months).

Just fyi, I also have to approve the patent application, but I typically review it at its final stage (after you have seen it and it is technically correct).

Take care,

Joe

From: Walker, William J (William)
Sent: Wednesday, January 26, 2000 8:38 AM
To: Opalach, Joseph J (Joe)
Cc: Huang, Yean-Ming (Ming); Bauer, Eric (Eric); Chien, Anthony H (Anthony); Walker, William J (William)
Subject: patent candidate

Joe -

My manager, Yean-Ming Huang asked me to forward this to you as
a candidate for a patent. It is a **Java to Xml Conversion** mechanism.
This has been developed for use in IP Exchange Comm for database
import/export and possibly data exchange. It is applicable to any Java/XML application. I have not been able to find any existing mechanisms that worked in this way.

Please let me know if you need any additional information,

Thanks,
-Bill Walker

<<File: XmlUtil.doc>>

Bill Walker
Lucent Technologies
732-817-4809
wjw@lucent.com

Lucent Technologies
Bell Labs Innovations

BELL LABORATORIES

Subject: **Patent Submission IDS #121717**
"A Mechanism For Converting Between Java Classes And XML"

Date: **January 31, 2000**

From: **Joseph J. Opalach**
Intellectual Property-Law
HO 3K-238 (732) 949-1708

B. J. Allain:

Patent submission # 121717 was formally docketed to consider the patentability of the above-identified subject matter. W. J. Walker appears to be the originator.

Should you have any questions regarding the subject matter, please feel free to contact me.

Joseph J. Opalach

Joseph J. Opalach
Corporate Counsel

HO-P33A70000-JJO-cl

Copy to:
W. J. Walker
A. H. Chien
Y. Huang

From: Opalach, Joseph J (Joe)
Sent: Thursday, January 27, 2000 11:12
To: Walker, William J (William)
Subject: RE: patent candidate

Hi Bill,

I've looked over your submission and we will move forward with it.

Most of the patent application development is contracted to outside attorneys, so I will get your submission rolling (so-to-speak) to get an outside attorney assigned. I will still be your contact within Lucent if you have any questions, complaints, etc.

The outside attorney should be in touch with you in approx. 2 to 3 months. I am going to "schedule" this submission for filing in 6 months (the filing date in the end has to do with the current workload of the assigned attorney and yourself, since you will have to provide assistance and review the patent application, so it could be filed sooner than 6 months).

Just fyi, I also have to approve the patent application, but I typically review it at its final stage (after you have seen it and it is technically correct).

Take care,

Joe

From: Walker, William J (William)
Sent: Wednesday, January 26, 2000 8:38 AM
To: Opalach, Joseph J (Joe)
Cc: Huang, Yean-Ming (Ming); Bauer, Eric (Eric); Chien, Anthony H (Anthony); Walker, William J (William)
Subject: patent candidate

Joe -

My manager, Yean-Ming Huang asked me to forward this to you as
a candidate for a patent. It is a **Java to Xml Conversion** mechanism.
This has been developed for use in IP Exchange Comm for database
import/export and possibly data exchange. It is applicable to any Java/XML application. I have not been able to find any existing mechanisms that worked in this way.

Please let me know if you need any additional information,

Thanks,
-Bill Walker

<<File: XmlUtil.doc>>

Bill Walker
Lucent Technologies
732-817-4809
wjw@lucent.com

Lucent Technologies
Bell Labs Innovations

BELL LABORATORIES

Subject: **Patent Submission IDS #121717**
"A Mechanism For Converting Between Java Classes And XML"

Date: **January 31, 2000**

From: **Joseph J. Opalach**
Intellectual Property-Law
HO 3K-238 (732) 949-1708

B. J. Allain:

Patent submission # 121717 was formally docketed to consider the patentability of the above-identified subject matter. W. J. Walker appears to be the originator.

Should you have any questions regarding the subject matter, please feel free to contact me.

Joseph J. Opalach/lc

Joseph J. Opalach
Corporate Counsel

HO-P33A70000-JJO-cl

Copy to:
W. J. Walker
A. H. Chien
Y. Huang

SUBMIS[illegible] - OPEN, RE-OPEN, OR TRA[illegible]FER

G. Ranieri
General Attorney

August 3, 2000
Date

____OPEN Enter this information into the "mksub" database, together with Brief Description (if available), and attach printsub page to PT 360.sub.

Assign to:____________________

Submission No.__________ Work Project No.__________

Government Contract No.__________ BU(s) Code:__________

Title and Name(s) to be used in the SUBJECT:

____RE-OPEN SUBMISSION__________

Assign to:__________ Effective Date:__________

Government Contract No.__________ BU(s) Code:__________

X TRANSFER SUBMISSION 121717

From: J. J. Opalach To: T. J. Bean Effective: 8/3/00

APPROVAL BY ORIGINATOR G. Ranieri /lc DATE: 8/3/00
General Attorney

APPROVAL IF ASSIGNED TO ANOTHER CENTER__________ DATE:__________
General Attorney

TO BE FILED IN SUBMISSION FOLDER

PT 360.sub 10/96

SUBMISSION INFORMA ON

Class Code: III

ATTORNEY: JJO

121717

SUBMISSION TITLE A MECHANISM FOR CONVERTING BETWEEN JAVA CLASSES AND XML

FILING DEADLINE: ______

DATE RECEIVED FROM INVENTOR: 1/27/00

INVENTOR: Walker William J.

Last	First	Middle

SSN (if known): ______
Organization No.: ______

INVENTOR: ______

Last	First	Middle

SSN (if known): ______
Organization No.: ______

INVENTOR: ______

Last	First	Middle

SSN (if known): ______
Organization No.: ______

INVENTOR: ______

Last	First	Middle

SSN (if known): ______
Organization No.: ______

INVENTOR: ______

Last	First	Middle

SSN (if known): ______
Organization No.: ______

INVENTOR: ______

Last	First	Middle

SSN (if known): ______
Organization No.: ______

Please attach all paperwork desired to be bound into folder

PT360.subinfo(10/97)

SUBMISSIONS/CASES REFERRED TO OUTSIDE COUNSEL

Legal Secretary

DATE ______

ATTORNEY JJD

IDS NUMBER 121717

CASE NAME/NUMBER ______

Provisional? ☐ Yes ☐ No

File Date ______

CLASSIFICATION CODE* III
[See NOTE #1 below]

SPECIALTY AREA/TECHNOLOGY DESIGNATION 2, C
[See Attachment A for Codes]

OFFICE ACTION DUE DATE** ______
[See NOTE #2 below]

FILING DEADLINE** ______
[To be completed for CRITICAL Dates ONLY]

OC FIRM PREFERENCE (IF ANY) ______

Office Manager

RECEIVED FROM LEGAL SECRETARY ______

SENT TO OUTSIDE COUNSEL GROUP ______

[illegible]

If designating a Class Code III, a foreign filing decision MUST be indicated:

FOREIGN FILING? ☐ Yes [✓ No]

-If Yes, MCC approval is required per the "Lucent Patent Filing and Maintenance Policy".

APPROVAL:

______ (MCC Signature) ______ Date

[illegible]

If OFFICE ACTION is past the due date set by the USPTO/Foreign Agents, or FILING DEADLINE is two months or less, please follow the "Expedited Outside Counsel Procedure".

SUBMISSION NO.	:	121717
ATTORNEY	:	Opalach, Joseph J
Title	:	

A Mechanism For Converting Between Java Classes And XML

------------------------------MAIN INFORMATION------------------------------

ITEM STATUS	: Open	LUCENT RATING	: III
STATUS DATE	: 1/31/00	GOVT. CONTRACT	: N
OPEN DATE	: 1/27/00	TYPE	: Patentability
CLOSE DATE	:	DEADLINE DATE	:
CLASS CODE	: III	TECHNOLOGY	:
BU CODES(S)	: GSGBM		

------------------------------SUBMITTER INFORMATION------------------------------

SUBMITTER NAME	:	Walker, William J
COMPANY	:	LUCENT
LOCATION	:	nj7460
EXTENSION	:	+1 732 817 4609
DEPARTMENT	:	13M2A0000
DIRECTOR	:	Brian J. Allain

Brief Description:

A Mechanism for Converting Between Java Classes and XML

Overview

Java and XML (eXtensible Markup Language) technologies provide developers with the tools to write portable code and operate on portable data. Support for XML in the Java platform is increasing with the availability of standard APIs and parsers such as SAX and DOM (Document Object Model). While these APIs provide standard mechanisms for reading XML documents, they work at a relatively low level. Using DOM for example, developers must have a detailed understanding of how to use the API to navigate nodes, elements and attributes to extract textual content, and then convert the text to useful program data types. This may be tedious, error prone, and requires the developer to work with classes outside the application domain.

This paper describes a mechanism that allows developers to convert easily between Java and XML representations of data while working exclusively with classes from the application domain. With very little work, developers can add XML support for complex hierarchies of any user defined class, Java primitives and wrapper classes (Integer, Boolean, Float, etc.) as well as collections of such objects.

A proposal by Sun Microsystems has been drafted that deals with this topic using a different approach *(An XML Data-Binding Facility for the Java Platform, Mark Reinhold, Core Java Platform Group, 30 July 1999 – see www.javasoft.com/xml)*. They propose a schema compiler that generates Java classes from an XML schema. The API proposed in this paper has the following advantages:

- No schema compiler is required to generate new Java classes.
- New or existing classes can be easily annotated to work with the API.
- The developer has full control and flexibility over how the classes get mapped to XML. Two totally different class implementations can work with the same XML representation in different ways.

The following sections discuss how the API is used including the API class descriptions.

An Example

Data is described in XML in a hierarchical way by tagging elements. An XML document contains one single element (the document element) which may contain any number of other elements. For example, a XML representation of a book store may be:

```
<bookStore>
   <name>The Programmer's Book Store</name>
   <address>
      <street>1 Industrial Way</street>
```

```
        <city>Middletown</city>
        <state>NJ</state>
        <zip>07701</zip>
      </address>
      <books>
        <book reviewed='no'>
          <title>Xml and Java</title>
          <author>Hiroshi Maruyama</author>
          <cost>$49.00</cost>
        </book>
        <book reviewed='yes'>
          <title>Java in a Nutshell</title>
          <author>Flannigan</author>
          <cost>$39.00</cost>
            <review>
            <reviewedBy>Joe</reviewedBy>
            <rating>3.5</rating>
          </review>
          <review>
            <reviewedBy>Bob</reviewedBy>
            <rating>9.5</rating>
          </review>
        </book>
      </books>
</bookStore>
```

The document contains a single element *bookStore*, which has sub-elements for *name*, *address*, and *books*. The *books* element contains a collection of *book* elements. Elements may contain one or more "attributes" such as the *reviewed* attribute of book, used to indicate whether the book has been reviewed or not. Books that have been reviewed contain one or more *review* elements.

This data can be modeled by a set of Java classes, such as:

```
class BookStore {
   String name;
   Address address;
   Vector books;

   BookStore(String name) {...}
   void setName(String name) {...}
}

class Book {
   String author;
   String title;
   float cost;

   Book(String title, String author);
   String getTitle() {...}
   String getAuthor() {...}
   ...
}

class ReviewedBook extends Book {
   Vector reviews;
   void addReview(Review review);
   Vector getReviews() { return reviews; }
}
```

The goal of the API is to let the developer construct the data using Java only, then somehow convert the classes to XML and later re-construct the contents of the Java classes from the XML file. For example:

```
BookStore bookStore = new BookStore("The Programmer's Book Store");
BookStore.setAddress( new Address("1 Maple St.", "Middletown", "NJ") );

Book book = new Book("Java in a Nutshell", "Flannigan");
Book.setCost(49.0f);
BookStore.add(book);

SaveToXml(bookStore, "bookStore.xml"); // a hypothetical method

// later, read the contents back to Java

BookStore bookStore = ReadFromXml("bookStore.xml"); // a hypothetical method
Collection books = bookStore.getBooks();
```

The next section describes the API and mechanism used to accomplishes this.

XML to Java API

The XmlReaderWriterInterface

In order to convert user-defined types to XML, such as *BookStore*, *Book*, and *Review* in the above example, each must include some instructions about how to do the conversion. The API needs to know:

- Which fields inside the Java object should be saved to XML? We may want the Java class to contain other fields that are used internally and should not get converted.
- For each field that we want converted, what tag name should be used when generating the corresponding XML element.
- When reading an XML file and constructing the java objects, what classes should be instantiated for each element? We want to be able to support different classes and different Java implementations using the same XML representation.

The Java to XML API accomplishes this by defining the *XmlReaderWriter* interface. Any class that we wish to convert to XML or construct from an XML document must implement this interface. The interface is defined as follows:

```
interface XmlReaderWriter {
   FieldDescription[] getFieldDescriptions();
   void setAttributes(Hashtable ht);
   Hashtable getAttributes();
}
```

The first method *getFieldDescriptions* is required. This allows the class to define how it should be converted. Typically, this will add just one line of code for all sub-elements contained by the class. The second two methods are optional, and are not used if the class does not use *attributes* as mentioned above. The *FieldDescription* class is described in the next section, but to illustrate it's use, the *BookStore* class from the last example can add the following lines of code to implement it:

```
class BookStore {

   public FieldDescription[] getFieldDescriptions() {
      return new FieldDescription[] {
         new FieldDescription("name", String.class, "getName", "setName"),
         new FieldDescription("address", Address.class, "getAddress", "setAddress"),
         new FieldDescription("books", Vector.class, "getBooks", "setBooks",
            Book.class, "book"),
      );
   }
   public void setAttributes(Hashtable ht) {;} // not used
   public void Hashtable getAttributes() { return null; } // not used
}
```

Note that the *BookStore* class only needs to describe fields directly contained in it. Sub-elements such as *Address* and *Book* (contained in the collection) will provide their own descriptions by implementing the interface.

The FieldDescription Class

The FieldDescription class provides the set of information needed by the API to convert between Java and XML representations. The FieldDescription class has the following constructors:

```
FieldDescription(String tagName, Class objectClass, String getMethod, String setMethod);

FieldDescription(String tagName, Class objectClass, String getMethod, String setMethod,
   Object contentClasses);

FieldDescription(String tagName, Class objectClass, String getMethod, String setMethod,
   Object contentClasses, Object ContantTagNames);
```

For simple elements, the first constructor is used. When a field is represented as a Collection, Hashtable or an array, the second forms are used to describe the elements in the collection.

For the first form, the parameters are:

- *TagName* – when writing this field to XML, what name should be used for the corresponding XML element tag.
- *ObjectClass* – Specifies the Class to instantiate when constructing this field from XML
- *GetMethod* – The name of the Java method to invoke to retrieve this field.
- *SetMethod* – The name of the Java method to invoke to retrieve this method.

The *contentClass* parameter is used to specify what class of object must be instantiated and constructed from the element. It may be any one of the following:

- A single Class object. In this case, all elements will be represented by the same class.
- The class may be based on element tag name. In this case, the parameter passed in is a Hashtable, where the keys are the element names and the values are the corresponding Class types.
- The class may be based on an attribute value contained in the elements. In this case, the parameter is a Hashtable where they keys are specified in the form *"attrName=attrValue"* and the values are the Class types to use.

A containing class must also specify the element names to use for each field when writing them to the XML document. The *contentName* parameter may be any of the following:

- The same name for all elements. In this case, pass a single String containing the name to use.
- The name may be obtained by invoking a "get" method on the object. In this case, you must specify a method name preceded by an "@" (e.g. "@getMyName"). This method must take no parameters and return a String.
- The name may be based on the class of object. In this case, the keys should be the Class types and the corresponding values should be the tag name to use.
- Based on Hashtable keys, if collection is an instance of java.util.Hashtable. In this case, use the FieldDescription constructor that does not take a contentName parameter.

In order to support inheritance, subclasses only need to define FieldDescriptions for new elements, and simply concatenate the FieldDescriptions of the parent class. For this purpose, the FieldDescription class has a *concat* method to make this easy. For example, the ReviewedBook class inherits from the Book class, so it's *getFieldDescription* method could be written as:

```
class ReviewedBook extends Book {
   ...
   public FieldDescription[] getFieldDescriptions() {
      FieldDescription[] fda = new FieldDescription[] {
         new FieldDescription("reviews", Vector.class, "getReviews", "setReviews",
               Review.class, "reviews")
      );
      return FieldDescription.concat( fda, super.getFieldDescriptions() );
   }
}
```

For the case of the book collection, we may just construct a Book object for each element if all we are interested in is the base class. However, if we want to construct a different type, depending on the attribute, we may do that as well. We can modify the *contentClass* parameter to be a Hashtable, and specify the type based on attribute:

```
Hashtable ht = new Hashtable();
ht.put("reviewed=yes", ReviewedBook.class);
ht.put("reviewed=no", Book.class);

fd = new FieldDescription("books", Vector.class, "getBooks", "setBooks",
            ht, "book");
```

Now the API can determine what type of class to instantiate based on attribute.

Attributes versus Elements

It is up to an implementation to decide whether to use elements or attributes. For example, consider a User object:

```
User {
   String id;
   String lastName;
   String firstName;
   String phoneNumber
}
```

In this object model, *id*, *lastName*, *firstName* and *phoneNumber* are considered "attributes" of User. In XML, this may be modeled as either:

```
<user>
   <id>1001</id>
   <lastName>Smith</lastName>
   <firstName>Joe</firstName>
   <phoneNumber>732-222-1234</phoneNumber>
</user>
```

or as

```
<user id='12345'>
   <lastName>Smith</lastName>
   <firstName>Joe</firstName>
   <phoneNumber>732-222-1234</phoneNumber>
</user>
```

For the second case, rather than describing *id* with a FieldDescription, it would be treated as an attribute in the User class:

```
Class user {
   String id = null;
   ..
   FieldDescription[] getFieldDescriptions() {
      return new FieldDescription[] {
         new FieldDescription("lastName', String.class, "getLastName', "setLastName'),
         new FieldDescription("firstName', String.class, "getFirstName', setFirstName'),
         new FieldDescription("phoneNumber', String.class, "getNumber', "setNumber'),
      };
   }
   Hashtable getAttributes() {
      Hashtable ht = new Hashtable();
      ht.put("id', id);
      return ht;
   }
   void setAttriutes(Hashtable ht) {
      String s = ht.get("id');
      if( s != null ) this.id = new String(s);
   }
}
```

The XmlUtil class

This class provides static methods that load and save Documents to and from XML streams as well as converting Document objects to and from specified Java classes. For the above book store example, the complete code needed to save the BookStore to an XML file and later restore it is as follows:

```
// construct a book store and populate it with books...
BookStore bookStore = new BookStore(...);
Book book = new Book(...);
BookStore.add(book);  // etc.

// turn it into a Document object and save to an XML file
Document doc = XmlUtil.getDocument("bookStore', bookStore);
```

```
XmlUtil.writeXml(doc, "books.xml');

// later, reload the book store from XML
Document doc = XmlUtil.readXml("books.xml');
BookStore bookStore = (BookStore)XmlUtil.getObject(doc, BookStore.class);
Collection books = bookStore.getBooks(); // do something with books
```

The *readXml* and *writeXml* are used to read and write *Documents* to and from XML files (or more generally streams). The conversion from a Java object to XML is accomplished by:

```
Document XmlUtil.getDocument(String docName, Object obj);
```

As long as the top level object and contained objects implement *XmlReaderWriter*, the whole collection can be handled by this call. To convert a Document to any class implementing XmlReaderWriter, use:

```
Object XmlUtil.getObject(Document doc, Class objectClass);
```

An instance of *objectClass* will be instantiated and queried for it's *FieldDescriptions*. From that point the API can determine how to convert all nodes that it encounters. This works recursively through the whole document tree. As mentioned, different object classes can be used to produce different results.

Summary

The API described here is a simple yet powerful mechanism for converting between Java and XML representations. It can be easily applied to any application that is written in Java and is using XML as an external data exchange format. The mechanism will work with any XML parser that implements the standard W3C Document Object model. XML representations are easily converted directly into the Java objects and data types used by developers within their application.